package com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.featurestorev1

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.ml.featurestore.lib.EntityId

import com.twitter.ml.featurestore.lib.data.PredictionRecordAdapter

import com.twitter.ml.featurestore.lib.entity.EntityWithId

import com.twitter.ml.featurestore.lib.online.FeatureStoreRequest

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.feature.featurestorev1.BaseFeatureStoreV1CandidateFeature

import com.twitter.product\_mixer.core.feature.featurestorev1.FeatureStoreV1CandidateEntity

import com.twitter.product\_mixer.core.feature.featurestorev1.featurevalue.FeatureStoreV1Response

import com.twitter.product\_mixer.core.feature.featurestorev1.featurevalue.FeatureStoreV1ResponseFeature

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.BaseBulkCandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.FeatureHydrationFailed

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import com.twitter.stitch.Stitch

import com.twitter.util.logging.Logging

trait FeatureStoreV1CandidateFeatureHydrator[

Query <: PipelineQuery,

Candidate <: UniversalNoun[Any]]

extends BaseBulkCandidateFeatureHydrator[

Query,

Candidate,

BaseFeatureStoreV1CandidateFeature[Query, Candidate, \_ <: EntityId, \_]

]

with Logging {

override def features: Set[BaseFeatureStoreV1CandidateFeature[Query, Candidate, \_ <: EntityId, \_]]

def clientBuilder: FeatureStoreV1DynamicClientBuilder

private lazy val hydrationConfig = FeatureStoreV1CandidateFeatureHydrationConfig(features)

private lazy val client = clientBuilder.build(hydrationConfig)

private lazy val datasetToFeatures =

FeatureStoreDatasetErrorHandler.datasetToFeaturesMapping(features)

private lazy val dataRecordAdapter =

PredictionRecordAdapter.oneToOne(hydrationConfig.allBoundFeatures)

private lazy val featureContext = hydrationConfig.allBoundFeatures.toFeatureContext

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[Candidate]]

): Stitch[Seq[FeatureMap]] = {

// Duplicate entities are expected across features, so de-dupe via the Set before converting to Seq

val entities: Seq[FeatureStoreV1CandidateEntity[Query, Candidate, \_ <: EntityId]] =

features.map(\_.entity).toSeq

val featureStoreRequests = candidates.map { candidate =>

val candidateEntityIds: Seq[EntityWithId[\_ <: EntityId]] =

entities.map(\_.entityWithId(query, candidate.candidate, candidate.features))

FeatureStoreRequest(entityIds = candidateEntityIds)

}

val featureMaps = client(featureStoreRequests, query).map { predictionRecords =>

if (predictionRecords.size == candidates.size)

predictionRecords

.zip(candidates).map {

case (predictionRecord, candidate) =>

val datasetErrors = predictionRecord.getDatasetHydrationErrors

val errorMap =

FeatureStoreDatasetErrorHandler.featureToHydrationErrors(

datasetToFeatures,

datasetErrors)

if (errorMap.nonEmpty) {

logger.debug(() =>

s"$identifier hydration errors for candidate ${candidate.candidate.id}: $errorMap")

}

val dataRecord =

new SRichDataRecord(

dataRecordAdapter.adaptToDataRecord(predictionRecord),

featureContext)

val featureStoreResponse =

FeatureStoreV1Response(dataRecord, errorMap)

FeatureMapBuilder()

.add(FeatureStoreV1ResponseFeature, featureStoreResponse).build()

}

else

// Should not happen as FSv1 is guaranteed to return a prediction record per feature store request

throw PipelineFailure(

FeatureHydrationFailed,

"Unexpected response length from Feature Store V1 while hydrating candidate features")

}

Stitch.callFuture(featureMaps)

}

}